REST

Representational **S**tate **T**ransfer

Contraintes REST

1. Une architecture client-serveur composée de clients, de serveurs et de ressources
2. Des communications client-serveur stateless (sans état)
3. Des données qui peuvent être mises en mémoire cache
4. Une interface uniforme (API) entre les composants qui permet un transfert standardisé des informations
5. Un système où des couches hiérarchiques assurent la médiation dans les interactions entre le client et le serveur final
6. Du code à la demande qui permet au serveur d'étendre la fonctionnalité d'un client en transférant le code exécutable

Qui utilise REST ?

Utilisation

- Utilisé dans le développement d'applications orientées ressources (ROA) et orientées données (DOA)
- Les API respectant pleinement l'architecture REST sont dites RESTful

Ressources et URI

- **Les URI identifient une ressource à manipuler**
- Ces ressources peuvent être une collection d'autres ressources ou une ressource membre d'une collection

- L'URI `/collection/add` n'est, par exemple, pas valide car `add` n'est pas une ressource mais une action sur un ressource...
  - L'URI valide est donc `/collection`
  - L'action `add` sera représentée sous une autre forme

Représentation des ressources

La **représentation** (REpresentational) d'une ressource désigne **les données échangées** entre le client et le serveur pour cette ressource.

- GET ➔ Le serveur renvoie au client l'état d'une ressource (ses données)
- POST, PUT ➔ Le client envoie au serveur l'état d'une ressource (ses données)

- Le format des données échangées n'est pas imposé par REST
  - On peut utiliser JSON, YAML, XML, etc. pour l'échange des données

Articulation de REST et HTTP

- Les ressources à manipuler sont identifiées par des URIs
- Le **type d'action** qu'on veut effectuer sur la ressource (consultation, ajout, suppression, ...) correspondent aux **méthodes HTTP** (GET, POST, DELETE, ...)
- Le **contexte** (authentification de l'utilisateur, ...) est envoyé dans les **headers**

CRUD

Acronyme désignant les quatre opérations fondamentales sur les données.

- **C**reate (POST)
- **R**ead (GET)
- **U**pdate (PUT/PATCH)
- **D**elete (DELETE)

Dans REST, chaque opération correspond à une méthode du protocole HTTP

Certains parlent même de **S**CRUD. S correspondant à Search (rechercher) qui est aussi une opération courante en informatique.

Endpoint

Un **endpoint** (ou point de terminaison) est la combinaison d'une URI et d'une méthode HTTP.

```
GET /users/{id}
POST /users/{id}/books
DELETE /books/{id}
...
```

À ne surtout pas faire ! (exemple)

1. Je veux donner la possibilité d'ajouter un utilisateur via mon API.
2. J'expose l'URI `monapi.com/users/add`, accessible via la méthode POST et attendant un body contenant les informations de mon nouvel utilisateur.

NON !

`/users/add` **n'est pas une ressource !**

Bonne pratique (exemple)

- Exposer la collection comme un ressource sur l'URI `monapi.com/users`
- Donner la possibilité d'ajouter un utilisateur via un POST sur cette URI

```
POST monapi.com/users
```

Être RESTful, ça veut dire quoi ?

- Une API dite RESTful respecte **toutes** les contraintes REST et exigences des protocoles utilisées pour construire une telle API
- REST est le nom et RESTful l'adjectif

- Une API ne respectant pas tous les principes REST mais se basant sur certains d'entre eux pourra être appelée « API REST » ou « API basée sur REST », mais pas RESTful...

Comment ne pas être RESTful...

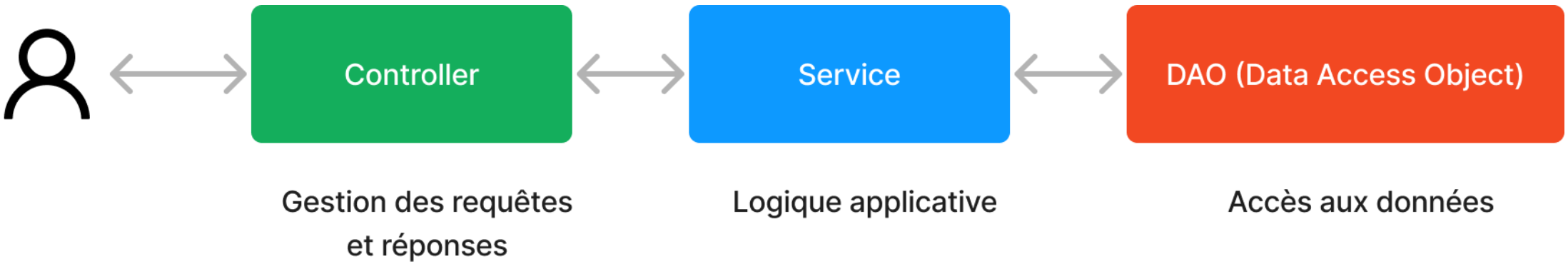
- Ne pas implémenter ou utiliser un système de cache
- Effectuer une lecture/consultation d'une ressource en POST
- Supprimer une ressource en GET
- ...

REST n'interdit pas ces options... MAIS si vous le faites, votre API ne respecte pas les exigences REST et n'est donc pas RESTful

Développement

Techniquement, ça donne quoi ?

Plusieurs couches de traitement



DAO et DTO

- Data Access Object
  - Objet destiné à l'accès aux données (dans une base de données par exemple)
- Data Transfer Object
  - Objet destiné aux transfert des données (entrée et sortie du controller)

Quelques frameworks

1. Express (NodeJS) https://expressjs.com/fr/
2. Django REST framework (Python) https://www.django-rest-framework.org/
3. Ruby on Rails (Ruby) https://rubyonrails.org/